# 18-函数式编程之组合性：函数式编程为什么如此吸引人？

你好！我是郑晔。

从上一讲开始，我们开启了函数式编程之旅，相信你已经对函数式编程有了一个初步的认识。函数式编程是一种以函数为编程元素的编程范式。但是，如果只有函数这一样东西，即使是说出花来，也没有什么特别的地方。

之前我讲过，GC来自于函数式编程，Lambda也来自于函数式编程。此外，在 Java 8增加的对函数式编程的处理中，流（Stream）的概念也从函数式编程中来，Optional也和函数式编程中的一些概念有着紧密的联系。由此可见，函数式编程给我们提供了许多优秀的内容。

接下来，我们来**讲讲函数式编程在设计上对我们帮助最大的两个特性：组合性和不变性。**

首先，我们来讨论一下组合性，看看函数式编程为什么能够如此吸引人。

## 组合行为的高阶函数

在函数式编程中，有一类比较特殊的函数，它们可以接收函数作为输入，或者返回一个函数作为输出。这种函数叫做**高阶函数**（High-order function）。

听上去稍微有点复杂，如果我们回想一下高中数学里有一个复合函数的概念，也就是 f(g(x)) ，把一个函数和另一个函数组合起来，这么一类比，是不是就好接受一点了。

那么，**高阶函数有什么用呢？它的一个重要作用在于，我们可以用它去做行为的组合**。我们再来回顾一下上一讲写过的一段代码：

```
find(byName(name).and(bySno(sno)));
```

在这里面，find的方法就扮演了一个高阶函数的角色。它接收了一个函数作为参数，由此，一些处理逻辑就可以外置出去。这段代码的使用者，就可以按照自己的需要任意组合。

你可能注意到了，这里的find方法只是一个普通的Java函数。是这样的，如果不需要把这个函数传来传去，普通的Java函数也可以扮演高阶函数的角色。

可以这么说，高阶函数的出现，让程序的编写方式出现了质变。按照传统的方式，程序库的提供者要提供一个又一个的完整功能，就像findByNameAndBySno这样，但按照函数式编程的理念，提供者提供的就变成了一个又一个的构造块，像find、byName、bySno这样。然后，使用者可以根据自己的需要进行组合，非常灵活，甚至可以创造出我们未曾想过的组合方式。

这就是典型的函数式编程风格。**模型提供者提供出来的是一个又一个的构造块，以及它们的组合方式。由使用者根据自己需要将这些构造块组合起来，提供出新的模型，供其他开发者使用**。就这样，模型之间一层又一层地逐步叠加，最终构建起我们的整个应用。

前面我们讲过，一个好模型的设计就是逐层叠加。**函数式编程的组合性，就是一种好的设计方式**。

但是，能把模型拆解成多个可以组合的构造块，这个过程非常考验人的洞察力，也是“分离关注点”的能力，但是这个过程可以让人得到一种智力上的愉悦。为什么函数式编程一直处于整个IT行业的角落里，还能吸引一大批优秀的开发者前赴后继地投入其中呢？这种智力上的愉悦就是一个重要的原因。

还记得我们在课程一开始讲的分层模型吗？这一点在函数式编程社区得到了非常好的体现。著名的创业孵化器Y Combinator的创始人Paul Graham曾经写过一篇文章《The Roots of Lisp》（中文版），其中用了七个原始操作符加上函数定义的方式，构建起一门LISP语言。

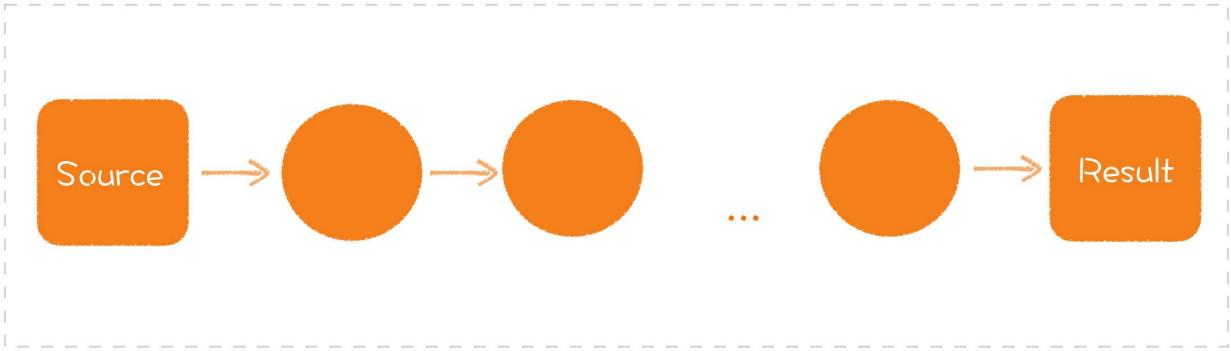
没错，是构建了一门语言。有了语言，你就可以去完成任何你想做的事了。这篇文章非常好地体现了函数式编程社区这种逐步叠加构建模型的思想。有兴趣的话，你可以去读一下。

当我们把模型拆解成小的构造块，如果构造块足够小，我们自然就会发现一些通用的构造块。

### 列表转换思维

我们说过，早期的函数式编程探索是从LISP语言开始的。LISP这个名字源自“List Processing”，这个名字指明了这个语言中的一个核心概念：List，也就是列表。程序员对List并不陌生，这是一种最为常用的数据结构，现在的程序语言几乎都提供了各自List的实现。

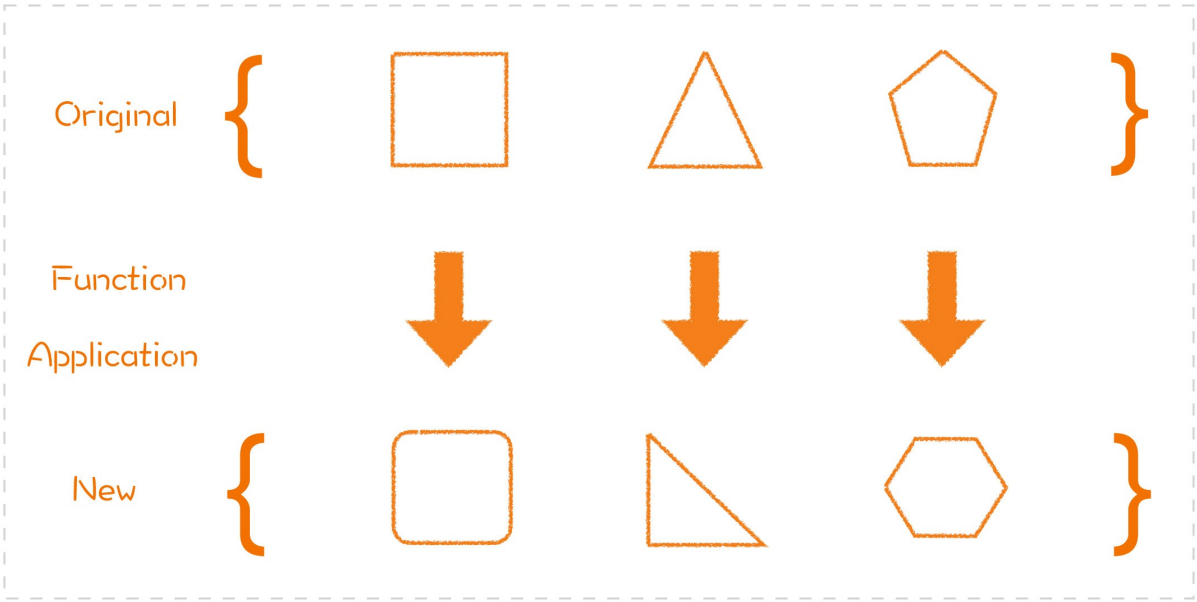
LISP 的一个洞见就是，大部分操作最后都可以归结成列表转换，也就是说，数据经过一系列的列表转换会得到一个结果，如下图所示：




**想要理解这一系列的转换，就要先对每个基础的转换有所了解。最基础的列表转换有三种典型模式，分别是map、filter和reduce**。如果我们能够正确理解它们，基本上就可以把for循环抛之脑后了。做过大数据相关工作的同学一定听说过一个概念：MapReduce，这是最早的一个大数据处理框架，这里的map和reduce就是源自函数式编程里列表转换的模式。

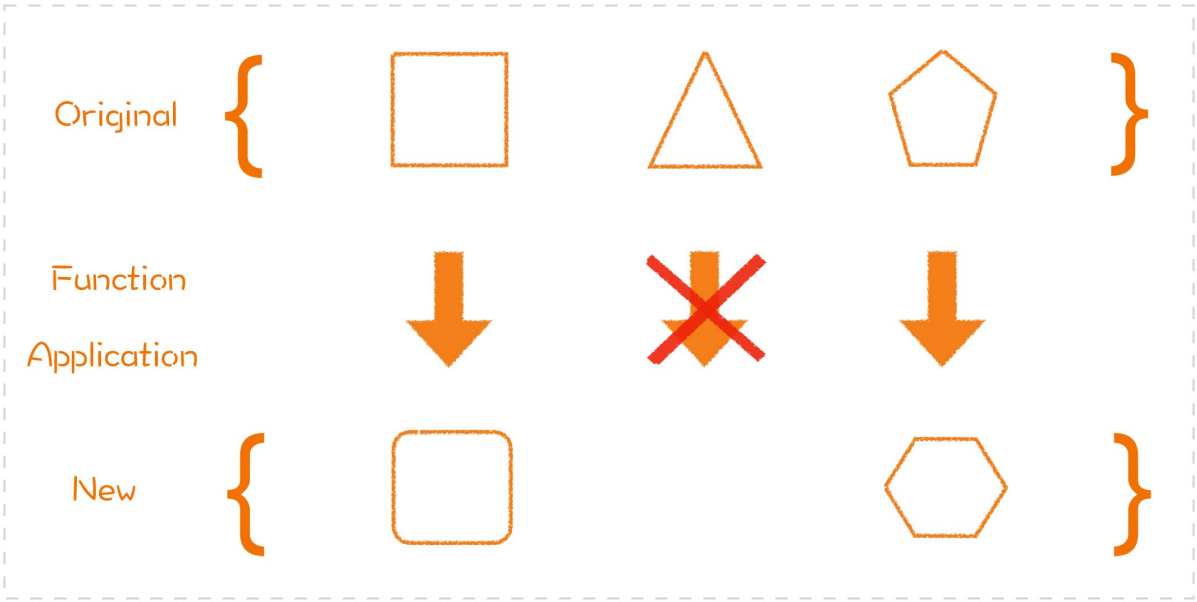
接下来，我们就来一个一个地看看它们分别是什么。

首先是map。map就是把一组数据通过一个函数映射为另一组数据。



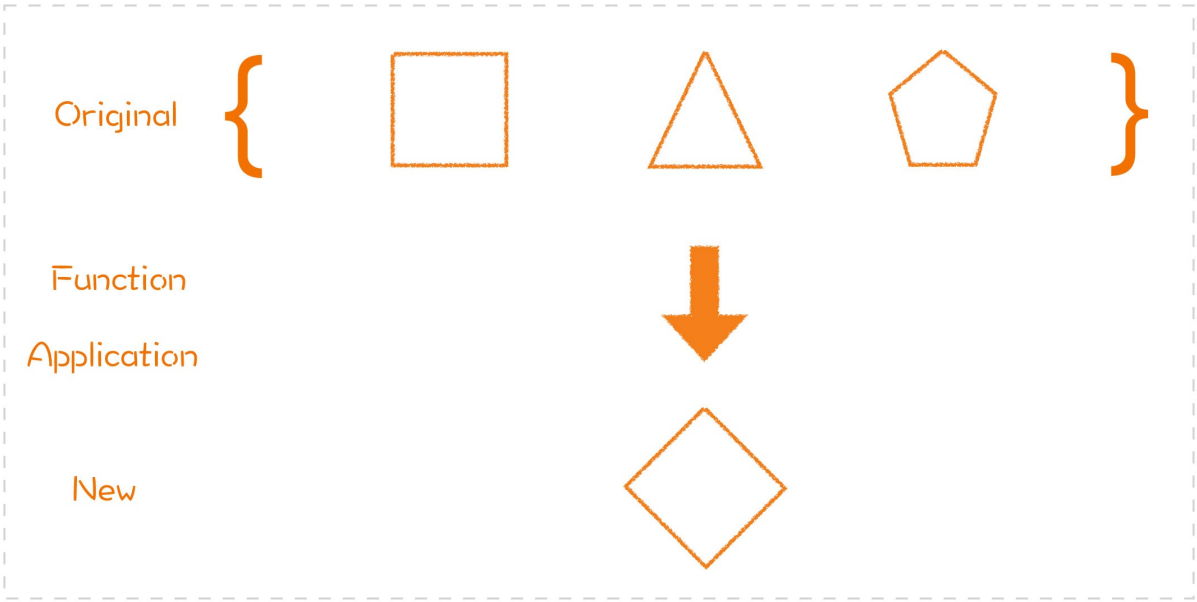

比如，我有一组数[1、2、3、4]，然后做了一个map操作，这里用作映射的函数是乘以2，也就是说，这组数里面的每个元素都乘以2，这样，我就得到了一组新的数[2、4、6、8]。

再来看filter。filter是把一组数据按照某个条件进行过滤，只有满足条件的数据才会留下。




同样[1、2、3、4]为例，我们做一个filter操作，过滤的函数是大于2，也就是说，只有大于2的数才会留下，得到的结果就是[3、4]。

最后是reduce。reduce就是把一组数据按照某个规则，归约为一个数据。




还是[1、2、3、4]，如果我们做一个reduce操作，其归约函数是一个加法操作，也就是这组数里面的每个元素相加，最终会得到一个结果，也就是 1+2+3+4=10。

好，有了基础之后，我们就可以利用这些最基础的转换模式去尝试解决问题了。比如，上一讲我们讲了一个学生的例子，现在，我们想知道这些学生里男生的总数。我们可以给Student类增加一个性别的字段：

```
// 单个学生的定义
class Student {
  ...
  // 性别
  private Gender gender;
}
```

要想知道男生的总数，传统做法应该是这么做：

```
long countMale() {
  long count = 0;
  for (Student student : students) {
    if (Gender.MALE == student.getGender())) {
        count++;
    }
  }

  return count;
}
```

按照列表转换的思维来做的话，我们该怎么做呢？首先，要把这个过程做一个分解：

- 取出性别字段；

- 判别性别是否为男性；
- 计数加1。

这三步刚好对应着map、filter和reduce：

- 取出性别字段，对应着map，其映射函数是取出学生的性别字段；
- 判别性别是否为男性，对应filter，其过滤函数是，性别为男性；
- 计数加1，对应着reduce，其归约函数是，加1。

有了这个分解的结果，我们再把它映射到代码上。Java 8对于函数式编程的支持，除了Lambda之外，它也增加了对列表转换的支持。为了兼容原有的API，它提供了一个新的接口：Stream，你可以把它理解成List的另一种表现形式。如果把上面的步骤用Java 8的Stream方式写出来，代码应该是这样的：

```
long countMale() {
    return students.stream()
            .map(student -> student.getGender())
            .filter(gender -> gender == Gender.MALE)
            .map(gender -> 1L)
            .reduce(0L, (sum, element) -> sum + element);
}
```

这基本和上面操作步骤是一一对应的，只是多了一步将性别转换成1，便于后面的计算。

map、filter和reduce只是最基础的三个操作，列表转换可以提供的操作远远比这个要多。不过，你可以这么理解，大多数都是在这三个基础上进行了封装，提供一种快捷方式。比如，上面代码的最后两步map和reduce，在Java 8的Stream接口提供了一个count方式，可以写成方法：

```
long countMale() {
    return students.stream()
            .map(Student::getGender)
            .filter(byGender(Gender.MALE))
            .count();
}

static Predicate<Gender> byGender(final Gender target) {
    return gender -> gender == target;
}
```

一方面，我用了方法引用（Student::getGender），这是Java提供的简化代码编写的一种方式。另一方面，我还把按照性别比较提取了出来，如此一来，代码的可读性就提升了，你基本上可以把它同前面写的操作步骤完全对应起来了。

同样是一组数据的处理，我更鼓励使用函数式的列表转换，而不是传统的 for 循环。一方面因为它是一种更有表达性的写法，从前面的代码就可以看到，它几乎和我们想做的事是一一对应的。另一方面，这里面提取出来比较性别的方法，它就是一个可以用作组合的基础接口，可以在多种场合复用。

很多Java程序员适应不了这种写法，一个重要的原因在于，他们缺少对于列表转换的理解。缺少了一个重要的中间环节，必然会出现不适。

你回想一下，我们说过结构化编程给我们提供了一些基础的控制结构，那其实也是一层封装，只不过，我们在编程之初就熟悉了if、for之类的写法。如果你同样熟悉函数式编程的基础设施，这些代码理解起来同那些控制结构没有什么本质区别，而且这些基础设施的抽象级别要比那些控制结构更高，提供了更好的表达性。

我们之前在讲DSL的时候就谈到过代码的表达性，其中一个重要的观点就是，有一个描述了做什么的接口之后，具体怎么做就可以在背后不断地进行优化。比如，如果一个列表的数据特别多，我们可以考虑采用并发的方式进行处理，而这种优化在使用端完全可以做到不可见。MapReduce 甚至将运算分散到不同的机器上执行，其背后的逻辑是一致的。

## 面向对象与函数式编程的组合

至此，我们已经学习了函数式编程的组合。你可能会有一个疑问，我们之前在讲面向对象的时候，也谈到了组合，这里讲函数式编程，又谈到了组合。这两种组合之间是什么关系呢？其实，对比一下代码，你就不难发现了，面向对象组合的元素是类和对象，而函数式编程组合的是函数。

这也就牵扯到在实际工作中，如何将面向对象和函数式编程两种不同的编程范式组合运用的问题。**我们可以用面向对象编程的方式对系统的结构进行搭建，然后，用函数式编程的理念对函数接口进行设计**。你可以把它理解成盖楼，用面向对象编程搭建大楼的骨架，用函数式编程设计门窗。

通过这两讲的例子，相信你已经感受到，一个好的函数式的接口，需要我们做的同样是“分离关注点”。虽然你不知道组合的方式会有多少种，但你知道，所有的变化其实就是一些基础元素的不断组合。在后面的巩固篇中，讲到Moco时，我们还会领略到这种函数式接口的魅力。

## 总结时刻

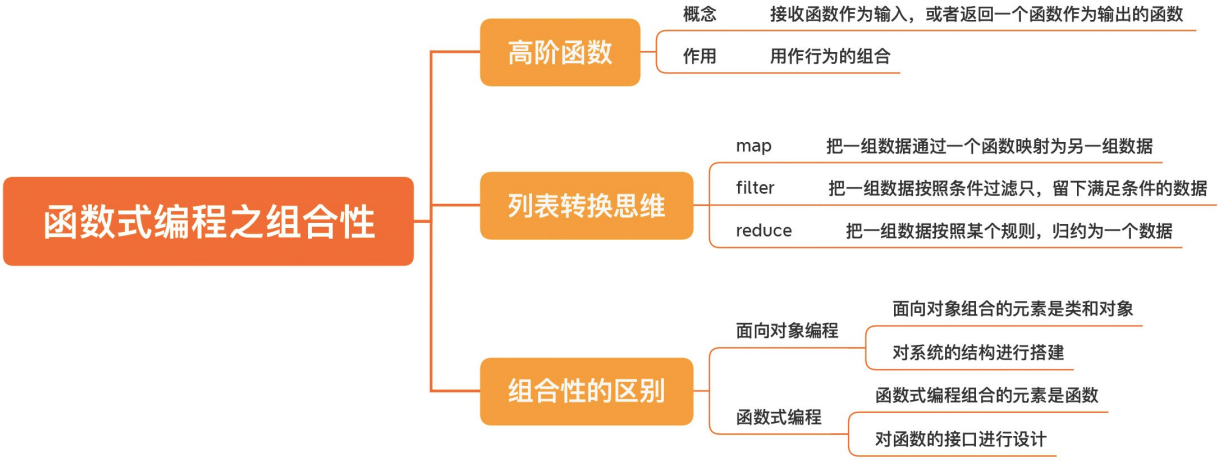
今天，我先给你讲了一类特殊的函数——高阶函数，它可以接受函数或返回函数。有了高阶函数，函数式编程就可以组合了，把不同的函数组合在一起完成功能，这也给逐层构建新抽象埋下了伏笔，函数式编程从此变得精彩起来。从设计的角度看，这种模型的层层叠加，是一种好的设计方式。

函数式编程中，还有一个重要的体系，就是列表转换的思想，将很多操作分解成若干转换的组合。最基础的三个转换是：map、filter和reduce，更多的转换操作都可以基于这三个转换完成。

面向对象和函数式编程都提到了组合性，不同的是，面向对象关键在于结构的组合，而函数式编程在于函数接口的组合。

组合性为我们提供了一个让函数接口组合的方式，下一讲我们再来讲一个让代码减少Bug的设计理念：不变性。

如果今天的内容你只能记住一件事，那请记住：**设计可以组合的函数接口**。




极客时间

## 思考题

函数式编程的组合性会给人带来极大的智力愉悦，你在学习软件开发的过程中，还有哪些东西曾经给你带来极大的智力愉悦呢？欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

## 精选留言：

- Being 2020-07-07 08:16:19
  大概就是通过拆解、组合的重构手法，减少for的圈复杂度吧，for嵌套多了确实头大。今天函数式编程的组合再次给了我启发，当我要写for循环的时候，就提醒自己可否用列表转化的思维尝试下。 [1赞]

  作者回复2020-07-07 10:53:54
  有这一点启发，足矣！

- J.D.Chi 2020-07-06 16:48:39
  之前看了一本书叫《函数式编程思维》，里面说了一个点，就是用函数式就是把一些事情交给编程语言去做，程序员不用去思考怎么实现，就像在list里的查找，我不用去考虑遍历的方法，只要给个条件，返回我要的结果就行了。 [1赞]

  作者回复2020-07-06 22:17:54
  这就是声明式编程，说明做什么，不必关心怎么做。

- Jxin 2020-07-06 01:31:54

  将单纯结构化的功能代码，重构成了领域模型+应用层引用的方式。属于领域模型的功能内敛，应用层对这些功能的复杂性无感。同时在多个应用层间，该领域模型的功能都是可以复用的，不管是代码去重还是复用性都有不错的提高。（让代码整洁合理，不确定是不是智力上的愉悦，但真的很爽。但不好的点是，烂代码的容忍度越来越差） [1赞]

  作者回复2020-07-06 11:42:09
  哈哈，烂代码容忍度越来越差，这不就是你水平提高了吗？

- NIU 2020-07-07 00:58:25
  map、filter 和 reduce 是编程语言的特性吧，在一些语言或语言的版本中不一定能看到？

作者回复2020-07-07 06:47:05
它们是一种常见的高阶函数。没有提供实现的语言也可以自己写一个，参考Guava中的实现。

- qinsi 2020-07-06 12:23:23
  js中仅使用解构操作实现列表及常用操作：https://exercism.io/tracks/javascript/exercises/list-ops/solutions/89919d0ba69743658f1ddb094a561b3e

- Geek_2e6a7e 2020-07-06 11:55:58
  函数式编程算是比较了解了，建议作者讲讲函数式学习的难点和思维方式，通过什么刻意练习提高？

- 阳仔 2020-07-06 00:52:36
  函数式编程其实也是一种对传统编程思想的转变
  面向对象编程是对系统结构的组合，函数式编程是对接口的组合
  这些编码规则或者范式也是对“分离关注点”的深刻理解之后抽象出来的标准模式
  如果同样问题经常出现，那么标准的解决方案也会出现